

```
CCCCCCCCCCCCC 000000000 NNN NNN VVV VVV
CCCCCCCCCCCCC 000000000 NNN NNN VVV VVV
CCCCCCCCCCCCC 000000000 NNN NNN VVV VVV
CCC 000 000 NNN NNN VVV VVV
CCC 000 000 NNN NNN VVV VVV
CCC 000 000 NNN NNN VVV VVV
CCC 000 000 NNNNNN NNN VVV VVV
CCC 000 000 NNNNNN NNN VVV VVV
CCC 000 000 NNNNNN NNN VVV VVV
CCC 000 000 NNN NNN NNN VVV VVV
CCC 000 000 NNN NNN NNN VVV VVV
CCC 000 000 NNN NNN NNN VVV VVV
CCC 000 000 NNN NNN NNN VVV VVV
CCC 000 000 NNN NNN NNN VVV VVV
CCC 000 000 NNN NNN NNN VVV VVV
CCC 000 000 NNN NNN NNN VVV VVV
CCCCCCCCCCCCC 000000000 NNN NNN VVV VVV
CCCCCCCCCCCCC 000000000 NNN NNN VVV VVV
CCCCCCCCCCCCC 000000000 NNN NNN VVV VVV
```

RRRRRRRR RRRRRRRR RR RR RR RR RRRRRRRR RRRRRRRR RR RR RR RR RR RR RR	EEEEEEEEEE EEEEEEEEEE EE EE EE EE EEEEEEEE EEEEEEEE EE EE EE EE EEEEEEEEEE EEEEEEEEEE	CCCCCCCC CCCCCCCC CC CC CC CC CC CC CC CC CC CC CCCCCCCC CCCCCCCC	LL LL LL LL LL LL LL LL LL LL LL LL LLLLLLLLLL LLLLLLLLLL	RRRRRRRR RRRRRRRR RR RR RR RR RRRRRRRR RRRRRRRR RR RR RR RR RR RR RR	EEEEEEEEEE EEEEEEEEEE EE EE EE EE EEEEEEEE EEEEEEEE EE EE EE EE EEEEEEEEEE EEEEEEEEEE	CCCCCCCC CCCCCCCC CC CC CC CC CC CC CC CC CC CC CCCCCCCC CCCCCCCC	.... .... .... ....
--	--	--	--	--	--	--	------------------------------

LL LL LL LL LL LL LL LL LL LL LL LLLLLLLLLL LLLLLLLLLL	IIIIII IIIIII II II II II II II II II II IIIIII IIIIII	SSSSSSSS SSSSSSSS SS SS SS SS SSSSSS SSSSSS SS SS SS SS SSSSSS SSSSSS
--	--	--



```
0001 0 %TITLE 'VAX-11 CONVERT/RECLAIM'
0002 0 MODULE RECL$REC ( IDENT='V04-000',
0003 0 OPTLEVEL=3
0004 0 ) =
0005 0
0006 1 BEGIN
0007 1
0008 1 *****
0009 1 *
0010 1 * COPYRIGHT (c) 1978, 1980, 1982, 1984 BY
0011 1 * DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.
0012 1 * ALL RIGHTS RESERVED.
0013 1 *
0014 1 * THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED
0015 1 * ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE AND WITH THE
0016 1 * INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR ANY OTHER
0017 1 * COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY
0018 1 * OTHER PERSON. NO TITLE TO AND OWNERSHIP OF THE SOFTWARE IS HEREBY
0019 1 * TRANSFERRED.
0020 1 *
0021 1 * THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT NOTICE
0022 1 * AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT
0023 1 * CORPORATION.
0024 1 *
0025 1 * DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF ITS
0026 1 * SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.
0027 1 *
0028 1 *
0029 1 *****
```

```
31 0030 1 ++
32 0031 1
33 0032 1 Facility: VAX-11 CONVERT/RECLAIM
34 0033 1
35 0034 1 Environment:
36 0035 1
37 0036 1 VAX/VMS Operating system
38 0037 1
39 0038 1 Abstract:
40 0039 1
41 0040 1 This module contains routines to handle index records.
42 0041 1
43 0042 1 Contents:
44 0043 1 BUCKET_EMPTY
45 0044 1 SQUISH_PRIMARY_BUCKET
46 0045 1 SQUISH_SIDR_BUCKET
47 0046 1 GET_DOWN_POINTER
48 0047 1 COMPARE_POINTER
49 0048 1 SWING_POINTER
50 0049 1 REMOVE_INDEX_RECORD
51 0050 1 RECOMPRESS_RECORD
52 0051 1
53 0052 1 --
54 0053 1
55 0054 1
56 0055 1 Author: Peter Lieberwirth Creation Date: 2-Sep-1981
57 0056 1
58 0057 1 Modified by:
59 0058 1
60 0059 1 V03-009 TMK0001 Todd M. Katz 03-Feb-1983
61 0060 1 Add support for Recovery Unit Journalling and RU ROLLBACK
62 0061 1 Recovery of ISAM files.
63 0062 1
64 0063 1 The routine SQUISH_PRIMARY_BUCKET has been modified to squish
65 0064 1 primary data records that are marked RU DELETE and re-format
66 0065 1 primary data records that have been marked RU_UPDATE.
67 0066 1
68 0067 1 The routine SQUISH_SIDR_BUCKET has been modified to squish
69 0068 1 SIDR array elements that are marked RU_DELETE.
70 0069 1
71 0070 1 NOTE: The routine SQUISH_SIDR_BUCKET is algorithmically wrong.
72 0071 1 It doesn't squish out anything! I plan on leaving it the way it
73 0072 1 is until a massive re-write can be done.
74 0073 1
75 0074 1 V03-008 KBT0396 Keith B. Thompson 2-Nov-1982
76 0075 1 Fix some bugs in squish_primary_bucket and squish_sidr_bucket
77 0076 1
78 0077 1 V03-007 KBT0389 Keith B. Thompson 28-Oct-1982
79 0078 1 Add support for prologue 3 sidrs and do record level
80 0079 1 space reclamation
81 0080 1
82 0081 1 V03-006 KBT0357 Keith B. Thompson 6-Oct-1982
83 0082 1 Use new merged ctx definitions
84 0083 1
85 0084 1 V03-005 KBT0354 Keith B. Thompson 5-Oct-1982
86 0085 1 Use new linkage definitions
87 0086 1
```



RECL\$REC  
V04-000

VAX-11 CONVERT/RECLAIM

G 12  
15-Sep-1984 23:59:42  
14-Sep-1984 12:14:05

VAX-11 Bliss-32 V4.0-742  
DISK\$VMSMASTER:[CONV.SRC]RECLREC.B32;1 Page 3 (2)

:	88	0087	1	:	V03-004	KBT0049	Keith Thompson	21-Apr-1982
:	89	0088	1	:			Add routine to check for last index record in bucket	
:	90	0089	1	:				
:	91	0090	1	:	V03-003	KBT0046	Keith Thompson	12-Apr-1982
:	92	0091	1	:			Fix compression bug and increase the key buffers to 257 bytes	
:	93	0092	1	:				
:	94	0093	1	:	V03-002	KBT0042	Keith Thompson	3-Apr-1982
:	95	0094	1	:			Add routines to compare and swing index pointers	
:	96	0095	1	:				
:	97	0096	1	:	V03-001	KBT0026	Keith Thompson	29-Mar-1982
:	98	0097	1	:			Do not reclaim data buckets with zero id	
:	99	0098	1	:				

RECL\$REC  
V04-000

VAX-11 CONVERT/RECLAIM

H 12  
15-Sep-1984 23:59:42  
14-Sep-1984 12:14:05

VAX-11 Bliss-32 V4.0-742  
DISK\$VMSMASTER:[CONV.SRC]RECLREC.B32;1 Page 4 (3)

```
: 101      0099 1
: 102      0100 1 LIBRARY 'SYSSLIBRARY:LIB.L32';
: 103      0101 1 LIBRARY 'SRC$:CONVERT';
: 104      0102 1
: 105      0103 1 EXTERNAL ROUTINE
: 106      0104 1     CONV$SRMS_READ_ERROR      : NOVALUE;
: 107      0105 1
: 108      0106 1 FORWARD ROUTINE
: 109      0107 1     SQUISH_PRIMARY_BUCKET      : RL$JSB_REG_9 NOVALUE,
: 110      0108 1     SQUISH_SIDR_BUCKET         : RL$JSB_REG_9 NOVALUE,
: 111      0109 1     RECOMPRESS_RECORD          : RL$JSB_REG_8 NOVALUE;
: 112      0110 1
: 113      0111 1 EXTERNAL
: 114      0112 1     RECL$GL_BUCKET_COUNT,
: 115      0113 1     RECL$GL_SEARCH_BUFFER,
: 116      0114 1     CONV$AB_OUT_FAB             : $FAB_DECL,
: 117      0115 1     CONV$AB_OUT_RAB             : $RAB_DECL;
: 118      0116 1
: 119      0117 1 OWN
: 120      0118 1     INDEX,
: 121      0119 1     VBN_OFFSET,
: 122      0120 1     VBN_FREE_SPACE,
: 123      0121 1     KEY_BUFFER_1                :
: 124      0122 1     KEY_BUFFER_2                : BLOCK [ 257, BYTE ],
: 125      0123 1                                BLOCK [ 257, BYTE ];
```



```
127 0124 1 %SBTTL 'BUCKET_EMPTY'
128 0125 1 GLOBAL ROUTINE RECL$$BUCKET_EMPTY : RL$JSB_REG_9 =
129 0126 1 ++
130 0127 1
131 0128 1 Functional Description:
132 0129 1
133 0130 1 This routine determines if a bucket is empty. It handles both
134 0131 1 index level and data level buckets.
135 0132 1
136 0133 1 Calling Sequence:
137 0134 1
138 0135 1 RECL$$BUCKET_EMPTY();
139 0136 1
140 0137 1 Input Parameters:
141 0138 1
142 0139 1 None.
143 0140 1
144 0141 1 Implicit Inputs:
145 0142 1
146 0143 1 BUCKET - address of buffer containing bucket
147 0144 1
148 0145 1 Output Parameters:
149 0146 1
150 0147 1 None.
151 0148 1
152 0149 1 Implicit Outputs:
153 0150 1
154 0151 1 RECL$GL_BUCKET_COUNT is incremented.
155 0152 1
156 0153 1 Routine Value:
157 0154 1
158 0155 1 TRUE if bucket is empty
159 0156 1 FALSE if bucket is not empty or can't be reclaimed
160 0157 1
161 0158 1 Routines Called:
162 0159 1
163 0160 1 SQUISH_PRIMARY_BUCKET
164 0161 1 SQUISH_SIDR_BUCKET
165 0162 1
166 0163 1 Side Effects:
167 0164 1
168 0165 1 None.
169 0166 1
170 0167 1 --
171 0168 1
172 0169 2 BEGIN
173 0170 2
174 0171 2 DEFINE_CTX;
175 0172 2 DEFINE_BUCKET;
176 0173 2 DEFINE_KEY_DESC;
177 0174 2
178 0175 2 LITERAL
179 0176 2 RECL$DATA_LEVEL = 0;
180 0177 2 RECL$BUCKET_EMPTY = 1;
181 0178 2 RECL$BUCKET_NOT_EMPTY = 0;
182 0179 2
183 0180 2 ! Determine if bucket is data level or index level
```

```
184 0181 2 !
185 0182 2 ! IF .BUCKET [ BKT$B_LEVEL ] EQLU RECL$_DATA_LEVEL
186 0183 2 THEN
187 0184 2 ! Determine key
188 0185 2 !
189 0186 2 ! IF .KEY_DESC [ KEY$B_KEYREF ] EQL 0
190 0187 2 THEN
191 0188 2 SQUISH_PRIMARY_BUCKET()
192 0189 2 ELSE
193 0190 2 SQUISH_SIDR_BUCKET();
194 0191 2
195 0192 2 ! See if it's empty
196 0193 2 !
197 0194 2 ! NOTE: Never reclaim the last bucket in a level, due to the complexity of
198 0195 2 ! updating high key values in all the levels above. This is not a serious
199 0196 2 ! restriction since most reclamation will be of aging buckets early in
200 0197 2 ! collating sequence.
201 0198 2 !
202 0199 2 IF ( .BUCKET [ BKT$W_KEYFRESPACE ] NEQU BKT$C_OVERHDSZ ) OR
203 0200 2 .BUCKET [ BKT$V_LASTBKT ]
204 0201 2 THEN
205 0202 2 RETURN RECL$_BUCKET_NOT_EMPTY
206 0203 2 ELSE
207 0204 2 RETURN RECL$_BUCKET_EMPTY
208 0205 2
209 0206 2
210 0207 1 END;
```

.TITLE RECL\$REC VAX-11 CONVERT/RECLAIM  
.IDENT \V04-000\

.PSECT \$OWN\$,NOEXE,2

00000 INDEX: .BLKB 4  
00004 VBN\_OFFSET: .BLKB 4  
00008 VBN\_FREE\_SPACE: .BLKB 4  
0000C KEY\_BUFFER\_1: .BCKB 257  
0010D .BLKB 3  
00110 KEY\_BUFFER\_2: .BCKB 257

.EXTRN CONV\$SRMS\_READ\_ERROR  
.EXTRN RECL\$GL\_BUCKET\_COUNT  
.EXTRN RECL\$GL\_SEARCH\_BUFFER  
.EXTRN CONV\$AB\_OUT\_FAB  
.EXTRN CONV\$AB\_OUT\_RAB

.PSECT \$CODE\$,NOWRT,2

0C A9 95 00000 RECL\$\$BUCKET\_EMPTY::  
TSTB 12(BUCKET)  
15 0D 12 00003 BNEQ 25  
AB 95 00005 TSTB 21(KEY\_DESC)

: 0182  
:  
: 0187



RECL\$REC  
V04-000

VAX-11 CONVERT/RECLAIM  
BUCKET\_EMPTY

K 12  
15-Sep-1984 23:59:42  
14-Sep-1984 12:14:05

VAX-11 Bliss-32 V4.0-742  
DISK\$VMSMASTER:[CONV.SRC]RECLREC.B32;1 Page 7  
(4)

		05	12	00008	BNEQ	1\$	:	
		0000V	30	0000A	BSBW	SQUISH_PRIMARY_BUCKET	:	0189
		03	11	0000D	BRB	2\$	:	
		0000V	30	0000F	BSBW	SQUISH_SIDR_BUCKET	:	0191
0E	04	A9	B1	00012	CMPW	4(BUCKET), #14	:	0200
		04	12	00016	BNEQ	3\$	:	
03	0D	A9	E9	00018	BLBC	13(BUCKET), 4\$	:	0201
		50	D4	0001C	CLRL	R0	:	0205
			05	0001E	RSB		:	
50		01	D0	0001F	MOVL	#1, R0	:	
			05	00022	RSB		:	0207

; Routine Size: 35 bytes, Routine Base: \$CODE\$ + 0000

```
212 0208 1 %SBTTL 'SQUISH PRIMARY_BUCKET'
213 0209 1 ROUTINE SQUISH_PRIMARY_BUCKET : RL$JSB_REG_9 NOVALUE =
214 0210 1 ++
215 0211 1
216 0212 1 Functional Description:
217 0213 1
218 0214 1     Squishes the deleted records out of the primary data buckets
219 0215 1
220 0216 1 Calling Sequence:
221 0217 1
222 0218 1     SQUISH_PRIMARY_BUCKET()
223 0219 1
224 0220 1 Input Parameters:
225 0221 1     None
226 0222 1
227 0223 1 Implicit Inputs:
228 0224 1
229 0225 1     BUCKET          - address of buffer containing bucket
230 0226 1     KEY_DESC
231 0227 1
232 0228 1 Output Parameters:
233 0229 1     None
234 0230 1
235 0231 1 Implicit Outputs:
236 0232 1     None
237 0233 1
238 0234 1 Routine Value:
239 0235 1     None
240 0236 1
241 0237 1 Routines Called:
242 0238 1
243 0239 1     None.
244 0240 1
245 0241 1 Side Effects:
246 0242 1
247 0243 1     None.
248 0244 1
249 0245 1 --
250 0246 1
251 0247 2 BEGIN
252 0248 2
253 0249 2 DEFINE_BUCKET;
254 0250 2 DEFINE_KEY_DESC;
255 0251 2
256 0252 2 LOCAL
257 0253 2     LAST,
258 0254 2     POINTER,
259 0255 2     RECORD_CTRL      : REF BLOCK [ ,BYTE ];
260 0256 2
261 0257 2     ! Point to the first record in the bucket
262 0258 2     !
263 0259 2     POINTER = BKT$K_OVERHDSZ + .BUCKET;
264 0260 2
265 0261 2     LAST = .POINTER;
266 0262 2
267 0263 2     ! Count the bucket
268 0264 2     !
```



```
269 0265 2 RECL$GL_BUCKET_COUNT = .RECL$GL_BUCKET_COUNT + 1;
270 0266
271 0267 ! If this bucket has an id of zero then don't bother reclaiming it
272 0268
273 0269 IF .BUCKET [ BKT$W_NXTRECID ] EQLU 0
274 0270 THEN
275 0271 RETURN;
276 0272
277 0273 ! Loop untill we have looked at all of the records
278 0274
279 0275 WHILE .POINTER LSSU ( .BUCKET [ BKT$W_FREESPACE ] + .BUCKET )
280 0276 DO
281 0277 BEGIN
282 0278
283 0279 ! Point to the control bytes of the record
284 0280 RECORD_CTRL = .POINTER;
285 0281
286 0282
287 0283 ! If this record not deleted check to see if there were any deleted
288 0284 ! records before it, if so squish them out
289 0285
290 0286 IF NOT ( .RECORD_CTRL [ IRCSV_DELETED ]
291 0287 OR
292 0288 .RECORD_CTRL [ IRCSV_RU_DELETE ] )
293 0289 THEN
294 0290 BEGIN
295 0291
296 0292 LOCAL SQUISH;
297 0293
298 0294 ! The current record is not deleted so squish out the
299 0295 ! deleted ones if there where any
300 0296
301 0297 SQUISH = .POINTER - .LAST;
302 0298
303 0299 IF .SQUISH NEQ 0
304 0300 THEN
305 0301 BEGIN
306 0302
307 0303 LOCAL BYTES;
308 0304
309 0305 ! Number of bytes left in the bucket
310 0306
311 0307 BYTES = ( .BUCKET + .BUCKET [ BKT$W_FREESPACE ] ) - .POINTER;
312 0308
313 0309 ! Move the rest of the records
314 0310
315 0311 CH$MOVE( .BYTES, .POINTER, .LAST );
316 0312
317 0313 ! Update the bucket pointer
318 0314
319 0315 BUCKET [ BKT$W_FREESPACE ] = .BUCKET [ BKT$W_FREESPACE ] -
320 0316 .SQUISH;
321 0317
322 0318 ! Update our pointers
323 0319
324 0320 POINTER = .POINTER - .SQUISH;
325 0321 RECORD_CTRL = .POINTER
```

```
END;
: If the current non-deleted primary data record is marked RU_UPDATE
: then re-format at this time.
IF .RECORD_CTRL [ IRC$V_RU_UPDATE ]
THEN
BEGIN
LOCAL
BYTES,
FAKE_SIZE : WORD,
TRUE_SIZE : WORD;

: Turn of the RU_UPDATE bit and retrieve the record's true size
: and the number of bytes in the bucket it currently occupies.
RECORD_CTRL [ IRC$V_RU_UPDATE ] = 0;
FAKE_SIZE = .RECORD_CTRL [ 9,0,16,0 ];
TRUE_SIZE = (.RECORD_CTRL + .FAKE_SIZE + 9)<0,16>;

: Place the true size of the primary data record in the size
: field of the record overhead, shift the rest of the records
: in the bucket to take up the available space, and update the
: bucket's freespace offset pointer.
RECORD_CTRL [ 9,0,16,0 ] = .TRUE_SIZE;

BYTES = .BUCKET + .BUCKET [ BKT$W_FREESPACE ]
      - .RECORD_CTRL
      - .FAKE_SIZE;

IF .BYTES GTRU 0
THEN
CH$MOVE ( .BYTES,
          .RECORD_CTRL + .FAKE_SIZE,
          .RECORD_CTRL + .TRUE_SIZE );

BUCKET [ BKT$W_FREESPACE ] = .BUCKET [ BKT$W_FREESPACE ]
      - ( .FAKE_SIZE - .TRUE_SIZE );

END;
END;

: Find the next record
: Is this record a RRV record
IF .RECORD_CTRL [ IRC$V_RRV ]
THEN
: If this record has no RRV pointer then set the size to the
: smallest record there is
IF .RECORD_CTRL [ IRC$V_NOPTRSZ ]
THEN
```



```

383      ! The least case size of a record is 3 bytes (CTRL and ID)
384      !
385      POINTER = .POINTER + 3
386
387      ELSE
388
389      ! The size of the record with an RRV pointer is
390      ! CTRL, ID and Pointer Size (ID and VBN)
391      !
392      POINTER = .POINTER + 3 + .RECORD_CTRL [ IRC$V_PTRSZ ] + 4
393
394      ELSE
395
396      ! It is not a RRV, so does it have a size field
397      !
398      IF .KEY_DESC [ KEY$V_REC_COMPR ] OR
399      .KEY_DESC [ KEY$V_KEY_COMPR ] OR
400      ( .CONV$AB_OUT_FAB [ FAB$B_RFM ] EQL FAB$C_VAR )
401      THEN
402
403      ! Add the size of the record from the size field and control
404      !
405      POINTER = .POINTER + .RECORD_CTRL [ 9,0,16,0 ] + 11
406
407      ELSE
408
409      ! Add the size of the record and control bytes
410      !
411      POINTER = .POINTER + .CONV$AB_OUT_FAB [ FAB$W_MRS ] + 9;
412
413      ! If the last record was not deleted update the last record pointer
414      !
415      IF NOT ( .RECORD_CTRL [ IRC$V_DELETED ]
416      OR
417      .RECORD_CTRL [ IRC$V_RU_DELETE ] )
418      THEN
419      LAST = .POINTER
420
421      END;
422
423      ! Update the bucket pointer to catch the last record if it was deleted
424      !
425      ! We exit the loop under two cases, 1) the last n records were deleted
426      ! in which case LAST points to the first deleted record or 2) the last
427      ! record was not deleted in which case LAST will be pointing to the
428      ! END of the last record, i.e. same as freespace.
429      !
430      BUCKET [ BKT$W_FREESPACE ] = .LAST - .BUCKET;
431
432      RETURN
433
434      END;
```

			05FC	8F	BB	00000	SQUISH_PRIMARY_BUCKET:	PUSHR #M<R2,R3,R4,R5,R6,R7,R8,R10>	: 0209	
		5E		04	C2	00004		SUBL2 #4, SP	:	
		57		0E	A9	9E	00007	MOVAB 14(R9), POINTER	: 0259	
					57	DD	0000B	PUSHL POINTER	: 0261	
			0000G	CF	D6	0000D		INCL RECLSGL_BUCKET_COUNT	: 0265	
			06	A9	B5	00011		TSTW 6(BUCKET)	: 0269	
				03	12	00014		BNEQ 1\$	:	
				00C8	31	00016		BRW 14\$	:	
	04	AE		04	A9	9E	00019	1\$: MOVAB 4(BUCKET), 4(SP)	: 0275	
		51		04	BE	3C	0001E	2\$: MOVZWL @4(SP), R1	:	
50		51			59	C1	00022	ADDL3 BUCKET, R1, R0	:	
		50			57	D1	00026	CMPL POINTER, R0	:	
					03	1F	00029	BLSSU 3\$	:	
				00AE	31	0002B		BRW 13\$	:	
		56			57	DO	0002E	3\$: MOVL POINTER, RECORD_CTRL	: 0281	
	5A	66			02	E0	00031	BBS #2, (RECORD_CTRL), 6\$	: 0286	
	56	66			05	E0	00035	BBS #5, (RECORD_CTRL), 6\$	: 0288	
	58	57			6E	C3	00039	SUBL3 LAST, POINTER, SQUISH	: 0297	
					16	13	0003D	BEQL 4\$	: 0299	
	50	59			51	C1	0003F	ADDL3 R1, BUCKET, R0	: 0307	
		50			57	C2	00043	SUBL2 POINTER, BYTES	:	
00	BE	67			50	28	00046	MOVC3 BYTES, (POINTER), @LAST	: 0311	
		BE		04	58	A2	0004B	SUBW2 SQUISH, @4(SP)	: 0316	
		57			58	C2	0004F	SUBL2 SQUISH, POINTER	: 0320	
		56			57	DO	00052	MOVL POINTER, RECORD_CTRL	: 0321	
	36	66			06	E1	00055	4\$: BBC #6, (RECORD_CTRL), 6\$	: 0328	
		66			40	8F	8A	00059	BICB2 #64, (RECORD_CTRL)	: 0340
		51			09	A6	B0	0005D	MOVW 9(RECORD_CTRL), FAKE_SIZE	: 0341
		5A				51	3C	00061	MOVZWL FAKE_SIZE, R10	: 0342
	51	56			5A	C1	00064	ADDL3 R10, -RECORD_CTRL, R1	:	
		50			09	A1	B0	00068	MOVW 9(R1), TRUE_SIZE	:
		58			50	3C	0006C	MOVZWL TRUE_SIZE, R8	: 0349	
		A6		09	58	B0	0006F	MOVW R8, 9(RECORD_CTRL)	:	
		50			04	BE	3C	00073	MOVZWL @4(SP), R0	: 0351
		50			59	C0	00077	ADDL2 BUCKET, R0	:	
		50			56	C2	0007A	SUBL2 RECORD_CTRL, R0	: 0352	
		50			5A	C2	0007D	SUBL2 R10, BYTES	: 0353	
					05	13	00080	BEQL 5\$	: 0355	
	6846	61			50	28	00082	MOVC3 BYTES, (R1), (R8)[RECORD_CTRL]	: 0359	
	50	58			5A	C3	00087	5\$: SUBL3 R10, R8, R0	: 0362	
		BE		04	50	A0	0008B	ADWD2 R0, @4(SP)	:	
	15	66			03	E1	0008F	6\$: BBC #3, (RECORD_CTRL), 8\$	: 0370	
	05	66			04	E1	00093	BBC #4, (RECORD_CTRL), 7\$	: 0376	
		57			03	C0	00097	ADDL2 #3, POINTER	: 0381	
					32	11	0009A	BRB 11\$	:	
50		66			00	EF	0009C	7\$: EXTZV #0, #2, (RECORD_CTRL), R0	: 0388	
		57			07	A047	9E	000A1	MOVAB 7(R0)[POINTER], POINTER	:
					26	11	000A6	BRB 11\$	: 0376	
					10	AB	95	000AB	8\$: TSTB 16(KEY_DESC)	: 0394
					0C	19	000AB	BLSS 9\$	:	
	07	AB			06	E0	000AD	BBS #6, 16(KEY_DESC), 9\$	: 0395	
		02			0000G	CF	91	000B2	CMPB CONVSAB_OUT_FAB+31, #2	: 0396
					0B	12	000B7	BNEQ 10\$	:	
		50			09	A6	3C	00CB9	9\$: MOVZWL 9(RECORD_CTRL), R0	: 0401
		57			0B	A047	9E	000BD	MOVAB 11(R0)[POINTER], POINTER	:
					0A	11	000C2	BRB 11\$	:	



RECL\$REC  
V04-000

VAX-11 CONVERT/RECLAIM  
SQUISH\_PRIMARY\_BUCKET

D 13  
15-Sep-1984 23:59:42  
14-Sep-1984 12:14:05

VAX-11 Bliss-32 V4.0-742  
DISK\$VMSMASTER:[CONV.SRC]RECLREC.B32;1 Page 13  
(5)

		50	0000G	CF	3C	000C4	10\$:	MOVZWL	CONVSAB OUT FAB+54, R0	:	0407
		57	09	A047	9E	000C9		MOVAB	9(R0)[POINTER], POINTER	:	
	07	66		02	E0	000CE	11\$:	BBS	#2, (RECORD-CTRL), 12\$	:	0411
	03	66		05	E0	000D2		BBS	#5, (RECORD-CTRL), 12\$	:	0413
		6E		57	D0	000D6		MOVL	POINTER, LAST	:	0415
			FF42	31	000D9	12\$:	BRW	2\$		:	0411
	04	BE		59	A3	000DC	13\$:	SUBW3	BUCKET, LAST, @4(SP)	:	0426
		6E		08	C0	000E1	14\$:	ADDL2	#8, SP	:	0430
		5E	05FC	8F	BA	000E4		POPR	#^M<R2,R3,R4,R5,R6,R7,R8,R10>	:	
				05	000E8		RSB			:	

; Routine Size: 233 bytes,      Routine Base: \$CODE\$ + 0023



```
436 0431 1 %SBTTL 'SQUISH_SIDR_BUCKET'
437 0432 1 ROUTINE SQUISH_SIDR_BUCKET : RL$JSB_REG_9 NOVALUE =
438 0433 1 ++
439 0434 1
440 0435 1 Functional Description:
441 0436 1
442 0437 1     Squishes the deleted records out of the sidr data buckets
443 0438 1
444 0439 1 Calling Sequence:
445 0440 1
446 0441 1     SQUISH_SIDR_BUCKET()
447 0442 1
448 0443 1 Input Parameters:
449 0444 1     None
450 0445 1
451 0446 1 Implicit Inputs:
452 0447 1
453 0448 1     BUCKET          - address of buffer containing bucket
454 0449 1     KEY_DESC
455 0450 1
456 0451 1 Output Parameters:
457 0452 1     None
458 0453 1
459 0454 1 Implicit Outputs:
460 0455 1     None
461 0456 1
462 0457 1 Routine Value:
463 0458 1     None
464 0459 1
465 0460 1 Routines Called:
466 0461 1     None
467 0462 1
468 0463 1 Side Effects:
469 0464 1     None
470 0465 1
471 0466 1     NOTE: The routine SQUISH_SIDR_BUCKET is algorithmically wrong.
472 0467 1     It doesn't squish out anything! I plan on leaving it the way it
473 0468 1     is until a massive re-write can be done.
474 0469 1
475 0470 1 --
476 0471 2 BEGIN
477 0472 2
478 0473 2 DEFINE_BUCKET;
479 0474 2 DEFINE_KEY_DESC;
480 0475 2
481 0476 2 LOCAL
482 0477 2     LAST
483 0478 2     POINTER          : REF BLOCK [ ,BYTE ],
484 0479 2     SIDR              : REF BLOCK [ ,BYTE ];
485 0480 2
486 0481 2     ! Point to the first record in the bucket
487 0482 2     SIDR = BKT$K_OVERHDSZ + .BUCKET;
488 0483 2
489 0484 2     ! Count the bucket
490 0485 2
491 0486 2 RECL$GL_BUCKET_COUNT = .RECL$GL_BUCKET_COUNT + 1;
492 0487 2
```

```

493 0488
494 0489
495 0490
496 0491
497 0492
498 0493
499 0494
500 0495
501 0496
502 0497
503 0498
504 0499
505 0500
506 0501
507 0502
508 0503
509 0504
510 0505
511 0506
512 0507
513 0508
514 0509
515 0510
516 0511
517 0512
518 0513
519 0514
520 0515
521 0516
522 0517
523 0518
524 0519
525 0520
526 0521
527 0522
528 0523
529 0524
530 0525
531 0526
532 0527
533 0528
534 0529
535 0530
536 0531
537 0532
538 0533
539 0534
540 0535
541 0536
542 0537
543 0538
544 0539
545 0540
546 0541
547 0542
548 0543
549 0544

! Loop untill we have looked at all of the records
WHILE .SIDR LSSU ( .BUCKET [ BKT$W_FREESPACE ] + .BUCKET )
DO
  BEGIN
    ! Point to the first array element
    IF .KEY_DESC [ KEY$V_KEY_COMPR ]
    THEN
      POINTER = .SIDR + .SIDR [ 2,0,8,0 ] + 4
    ELSE
      POINTER = .SIDR + .KEY_DESC [ KEY$B_KEYSZ ] + 2;
    LAST = .POINTER;
    ! Loop untill we have looked at all of the array elements
    WHILE .POINTER LSSU ( .SIDR + .SIDR [ 0,0,16,0 ] + 2 )
    DO
      ! If this array element is deleted skip to the next one
      IF .POINTER [ IRC$V_DELETED ]
      OR
      .POINTER [ IRC$V_RU_DELETE ]
      THEN
        ! Is there a pointer
        IF .POINTER [ IRC$V_NOPTRSZ ]
        THEN
          POINTER = .POINTER + 1
        ELSE
          POINTER = .POINTER + 1 + .POINTER [ IRC$V_PTRSZ ] + 4
      ELSE
        BEGIN
          LOCAL SQUISH;
          ! The current sidr is not deleted so squish out the
          ! deleted ones if there where any
          SQUISH = .POINTER - .LAST;
          IF .SQUISH NEQ 0
          THEN
            BEGIN
              LOCAL BYTES;
              ! Number of bytes left in the bucket
              BYTES = ( .BUCKET + .BUCKET [ BKT$W_FREESPACE ] ) - .POINTER;
```



```
550      0545      5      ! Move the rest of the records
551      0546
552      0547      ! CH$MOVE( .BYTES,.POINTER,.LAST );
553      0548
554      0549      ! Update the bucket pointer
555      0550
556      0551      BUCKET [ BKT$W_FREESPACE ] = .BUCKET [ BKT$W_FREESPACE ] -
557      0552      ! .SQUISH;
558      0553
559      0554
560      0555      ! Update the sidr record size
561      0556
562      0557      SIDR [ 0,0,16,0 ] = .SIDR [ 0,0,16,0 ] - .SQUISH;
563      0558
564      0559      ! Update out pointers
565      0560
566      0561      POINTER = .POINTER - .SQUISH;
567      0562
568      0563      END;
569      0564
570      0565      ! Find the next sidr element
571      0566
572      0567      POINTER = .POINTER + 1 + .POINTER [ IRC$V_PTRSZ ] + 4;
573      0568
574      0569      LAST = .POINTER
575      0570
576      0571      END;
577      0572
578      0573      ! Is the sidr array completely deleted
579      0574
580      0575      IF .POINTER EQL ( .SIDR + .SIDR [ 0,0,16,0 ] )
581      0576      THEN
582      0577      BEGIN
583      0578
584      0579      ! Squish out the entire record (leaving SIDR pointing to the
585      0580      ! next sidr record)
586      0581
587      0582      CH$MOVE( .SIDR [ 0,0,16,0 ],.POINTER,.SIDR );
588      0583
589      0584      ! Update the bucket pointer
590      0585
591      0586      BUCKET [ BKT$W_FREESPACE ] = .BUCKET [ BKT$W_FREESPACE ] -
592      0587      ! .SIDR [ 0,0,16,0 ]
593      0588
594      0589      END
595      0590      ELSE
596      0591
597      0592      ! If we don't squish the record find the next one
598      0593
599      0594      SIDR = .SIDR + .SIDR [ 0,0,16,0 ] + 2
600      0595
601      0596      END;
602      0597
603      0598      RETURN
604      0599
605      0600      END;
```



			05FC	8F	BB	00000	SQUISH_SIDR_BUCKET:		
							PUSRR	#^M<R2,R3,R4,R5,R6,R7,R8,R10>	0432
		5E		04	C2	00004	SUBL2	#4, SP	
		56		A9	9E	00007	MOVAB	14(R9), SIDR	0483
			0000G	CF	D6	0000B	INCL	RECL\$GL_BUCKET_COUNT	0487
		5A		A9	9E	0000F	MOVAB	4(BUCKET), R10	0491
		50		6A	3C	00013	MOVZWL	(R10), R0	
		50		59	C0	00016	ADDL2	BUCKET, R0	
		50		56	D1	00019	CMPL	SIDR, R0	
				03	1F	0001C	BLSSU	2\$	
				0089	31	0001E	BRW	13\$	
	0B	10		06	E1	00021	BBC	#6, 16(KEY_DESC), 3\$	0497
		AB		A6	9A	00026	MOVZBL	2(SIDR), R0	0499
		50		04	A046	9E	MOVAB	4(R0)[SIDR], POINTER	
		57		09	11	0002F	BRB	4\$	
		50		14	AB	9A	MOVZBL	20(KEY_DESC), R0	0501
		57		02	A046	9E	MOVAB	2(R0)[SIDR], POINTER	
		6E		57	D0	0003A	MOVL	POINTER, LAST	0503
		50		66	3C	0003D	MOVZWL	(SIDR), R0	0507
		50		02	A046	9E	MOVAB	2(R0)[SIDR], R0	
		50		57	D1	00045	CMPL	POINTER, R0	
				45	1E	00048	BGEQU	10\$	
	04	67		02	E0	0004A	BBS	#2, (POINTER), 6\$	0512
	14	67		05	E1	0004E	BBC	#5, (POINTER), 8\$	0514
	04	67		04	E1	00052	BBC	#4, (POINTER), 7\$	0519
				57	D6	00056	INCL	POINTER	0521
				E3	11	00058	BRB	5\$	
50	67	02		00	EF	0005A	EXTZV	#0, #2, (POINTER), R0	0523
		57		05	A047	9E	MOVAB	5(R0)[POINTER], POINTER	
				D7	11	00064	BRB	5\$	0519
	58	57		6E	C3	00066	SUBL3	LAST, POINTER, SQUISH	0533
				17	13	0006A	BEQL	9\$	0535
		50		6A	3C	0006C	MOVZWL	(R10), R0	0543
		50		59	C0	0006F	ADDL2	BUCKET, R0	
		50		57	C2	00072	SUBL2	POINTER, BYTES	
	00	BE		50	28	00075	MOVC3	BYTES, (POINTER), @LAST	0547
		6A		58	A2	0007A	SUBW2	SQUISH, (R10)	0552
		66		58	A2	0007D	SUBW2	SQUISH, (SIDR)	0557
		57		58	C2	00080	SUBL2	SQUISH, POINTER	0561
50	67	02		00	EF	00083	EXTZV	#0, #2, (POINTER), R0	0567
		57		05	A047	9E	MOVAB	5(R0)[POINTER], POINTER	
				AB	11	0008D	BRB	4\$	0569
		50		66	3C	0008F	MOVZWL	(SIDR), R0	0575
		50		56	C0	00092	ADDL2	SIDR, R0	
		50		57	D1	00095	CMPL	POINTER, R0	
				09	12	00098	BNEQ	11\$	
	66	67		66	28	0009A	MOVC3	(SIDR), (POINTER), (SIDR)	0582
		6A		66	A2	0009E	SUBW2	(SIDR), (R10)	0587
				04	11	000A1	BRB	12\$	0586
		56		02	A0	9E	MOVAB	2(R0), SIDR	0594
				FF69	31	000A7	BRW	1\$	0575
		5E		04	C0	000AA	ADDL2	#4, SP	0600
			05FC	8F	BA	000AD	POPR	#^M<R2,R3,R4,R5,R6,R7,R8,R10>	

RECL\$REC  
V04-000

VAX-11 CONVERT/RECLAIM  
SQUISH\_SIDR\_BUCKET

I 13  
15-Sep-1984 23:59:42  
14-Sep-1984 12:14:05

VAX-11 Bliss-32 V4.0-742  
DISK\$VMSMASTER:[CONV.SRC]RECLREC.B32;1 Page 18 (6)

05 000B1

RSB

;

; Routine Size: 178 bytes, Routine Base: \$CODE\$ + 010C

```

: 607 0601 1 %SBTTL 'GET DOWN POINTER'
: 608 0602 1 GLOBAL ROUTINE RECL$GET_DOWN_POINTER ( VBN ) : RL$JSB_REG_8 =
: 609 0603 1 ++
: 610 0604 1
: 611 0605 1 Functional Description:
: 612 0606 1
: 613 0607 1 This routine searches the current buffer for the specified
: 614 0608 1 down pointer.
: 615 0609 1
: 616 0610 1 Calling Sequence:
: 617 0611 1
: 618 0612 1 GET_DOWN_POINTER( VBN );
: 619 0613 1
: 620 0614 1 Input Parameters:
: 621 0615 1
: 622 0616 1 VBN - VBN of bucket on level below being deleted
: 623 0617 1
: 624 0618 1 Implicit Inputs:
: 625 0619 1
: 626 0620 1 BUCKET - address of buffer containing bucket
: 627 0621 1 KEY_DESC
: 628 0622 1
: 629 0623 1 Output Parameters:
: 630 0624 1
: 631 0625 1 None.
: 632 0626 1
: 633 0627 1 Implicit Outputs:
: 634 0628 1
: 635 0629 1 If success:
: 636 0630 1
: 637 0631 1 INDEX - number of the index record to remove (0=first)
: 638 0632 1 KEY_POINTER - points to key part to delete
: 639 0633 1 KEY_BUFFER_1 - contains the expanded key bucket previous
: 640 0634 1 to one being deleted
: 641 0635 1 KEY_BUFFER_2 - contains the expanded key of one being deleted
: 642 0636 1
: 643 0637 1 If failure the contents of the above registers are undefined.
: 644 0638 1
: 645 0639 1 Routine Value:
: 646 0640 1
: 647 0641 1 TRUE if down pointer found, else FALSE
: 648 0642 1
: 649 0643 1 Routines Called:
: 650 0644 1
: 651 0645 1 None.
: 652 0646 1
: 653 0647 1 Side Effects:
: 654 0648 1
: 655 0649 1 None.
: 656 0650 1
: 657 0651 1 --
: 658 0652 1
: 659 0653 2 BEGIN
: 660 0654 2
: 661 0655 2 DEFINE_CTX;
: 662 0656 2 DEFINE_BUCKET;
: 663 0657 2 DEFINE_KEY_DESC;
```





```

      |
      |      ~ ilici ~
      |      ~~~~~~
      |      \      \ fill \
      |      ~~~~~~
key_buffer_1 | ilici : : :
      |      ~~~~~~
      |      ^
      |      |
      |      filled in when c=0 ( always the first key
      |      in the bucket )
      |
      | CH$COPY( src_len, src, fill, dst_len, dst )
      |
      | CH$COPY( .KEY_POINTER [ KEYSB_LENGTH ],
      |           .KEY_POINTER + 2,
      |           .( .KEY_POINTER + 1 +
      |               .KEY_POINTER [ KEYSB_LENGTH ] ),
      |           .KEY_DESC [ KEYSB_KEYSZ ] -
      |           .KEY_POINTER [ KEYSB_FRONT_COUNT ],
      |           KEY_BUFFER_1 + 2 +
      |           .KEY_POINTER [ KEYSB_FRONT_COUNT ] );
      |
      | Skip to the next key.
      | KEY_POINTER = .KEY_POINTER + 2 +
      |               .KEY_POINTER [ KEYSB_LENGTH ]
      |
      | END:
      |
      | Fill in key_buffer_2 with the expanded CURRENT key
      | first by stuffing the front compressed characters from
      | the previous key in key_buffer_1 then copy the rest
      | from the bucket extending it if rear truncation is present
      |
      | key_pointer
      | |
      | | ~~~~~~
      | | ilici ~~~~~~
      | | ~~~~~~
      | | \      \
      | | ~~~~~~
key_buffer_1 | ilici : : :
      | | ~~~~~~
      | | |      | fill \
      | | ~~~~~~
key_buffer_2 | ilici : : :
      | | ~~~~~~
      | | ^
      | | |
      | | filled in when from key_buffer_1
      | | or from .key_pointer when c=0
      |
      | Fill in the front if there were front compression
      |
      | CH$MOVE( .KEY_POINTER [ KEYSB_FRONT_COUNT ],
      |           KEY_BUFFER_1 + 2,
```



```

778      KEY_BUFFER_2 + 2);
779
780      ! Copy the rest of the key and expand the rear if neccessary
781      !
782      CH$COPY( .KEY_POINTER [ KEYSB_LENGTH ],
783              .KEY_POINTER + 2,
784              ( .KEY_POINTER + 1 + .KEY_POINTER [ KEYSB_LENGTH ] ),
785              .KEY_DESC [ KEYSB_KEYSZ ] =
786              .KEY_POINTER [ KEYSB_FRONT_COUNT ],
787              KEY_BUFFER_2 + 2 +
788              .KEY_POINTER [ KEYSB_FRONT_COUNT ] );
789
790      RETURN RECL$_SUCCESS
791
792      END
793
794      ELSE
795
796      BEGIN
797
798      ! The key is not compressed, so the key part is fixed length
799      ! and easy to find.
800
801      KEY_POINTER = ( .INDEX * .KEY_DESC[ KEYSB_KEYSZ ] )
802                  + .BUCKET + BRT$K_OVERHDSZ;
803
804      RETURN RECL$_SUCCESS
805
806      END
807
808      ELSE
809
810      BEGIN
811
812      ! This was not the down pointer, so get the next down pointer
813      !
814      VBN_OFFSET = .VBN_OFFSET - ( .BUCKET[ BKT$V_PTR_SZ ] + 2 );
815      INDEX = .INDEX + 1;
816
817      END;
818
819      ! If we fell through the UNTIL - DO loop it means we didn't find a down
820      ! pointer.
821      RETURN RECL$_FAILURE
822
823      END;
```

```

00FC 8F BB 0000 RECL$$GET_DOWN_POINTER::
5E      0C C2 00004      POSHR #^M<R2,R3,R4,R5,R6,R7>
      0000' CF D4 00007      SUBL2 #12, SP
56      58 AA 3C 0000B      CLRL INDEX
56      03 C2 0000F      MOVZWL 88(CTX), VBN_OFFSET
      SUBL2 #3, VBN_OFFSET
```

```

: 0602
: 0666
: 0671
:
```



RECL\$REC  
V04-000

VAX-11 CONVERT/RECLAIM  
GET\_DOWN\_POINTER

N 13  
15-Sep-1984 23:59:42  
14-Sep-1984 12:14:05

VAX-11 Bliss-32 V4.0-742  
DISK\$VMSMASTER:[CONV.SRC]RECLREC.B32;1 Page 23  
(7)

04	AE	0D	A9	08	AE	7649	9F	00012	PUSHAB	-(VBN_OFFSET)[BUCKET]	0675
			50	02	AE	9E	3C	00015	MOVZWL	4(SP), VBN_FREE_SPACE	0679
				56	04	03	EF	00019	EXTZV	#3, #2, 13(BUCKET), 4(SP)	0683
				56	FE	AE	C3	00020	SUBL3	4(SP), VBN_OFFSET, R0	0688
				08		A0	9E	00025	MOVAB	-2(R0), VBN_OFFSET	0689
						56	D1	00029	CMPL	VBN_OFFSET, -VBN_FREE_SPACE	0695
						03	1A	0002D	BGTRU	2\$	0703
						0090	31	0002F	BRW	8\$	0705
			50	04	AE	03	78	00032	ASHL	#3, 4(SP), R0	0728
51		6649		50	50	10	C0	00037	ADDL2	#16, R0	0730
				28	AE	00	EF	0003A	EXTZV	#0, R0, (VBN_OFFSET)[BUCKET], R1	0733
						51	D1	00040	CMPL	R1, VBN	0734
						6C	12	00044	BNEQ	7\$	0739
			54	10	AB	03	E1	00046	BBC	#3, 16(KEY_DESC), 5\$	0770
					58	0E	A9	0004B	MOVAB	14(R9), KEY_POINTER	0772
					6E	01	CE	0004F	MNEGL	#1, 1	0776
						21	11	00052	BRB	4\$	0780
						68	9A	00054	MOVZBL	(KEY_POINTER), R2	0781
			57			52	C1	00057	ADDL3	R2, KEY_POINTER, R7	0790
						50	A8	0005B	MOVZBL	1(KEY_POINTER), R0	0795
						51	AB	0005F	MOVZBL	20(KEY_DESC), R1	0796
51	01	A7	02	A8		50	C2	00063	SUBL2	R0, R1	0798
						52	2C	00066	MOVCS	R2, 2(KEY_POINTER), 1(R7), R1, -	0799
						0000'CF	40	0006D	MOVAB	2(R7), KEY_POINTER	0799
						02	A7	00071	AOBLS	INDEX, 1, 3\$	0799
						0000'	CF	00075	MOVZBL	1(KEY_POINTER), R7	0799
						01	A8	0007B	MOVCS	R7, KEY_BUFFER_1+2, KEY_BUFFER_2+2	0807
						57	28	0007F	MOVZBL	(KEY_POINTER), R0	0808
						50	68	00087	MOVZBL	20(KEY_DESC), R1	0815
						51	AB	0008A	SUBL2	R7, R1	0817
51	01	A048	02	A8		57	C2	0008E	MOVCS	R0, 2(KEY_POINTER), 1(R0)[KEY_POINTER], -	0817
						50	2C	00091	BRB	6\$	0796
						0000'CF	47	00099	MOVZBL	20(KEY_DESC), R0	0796
						0E	11	0009D	MULL2	INDEX, R0	0796
						14	AB	0009F	MOVAB	14(BUCKET)[R0], KEY_POINTER	0796
						0000'	CF	000A3	MOVL	#1, R0	0796
						0E	A940	000A8	BRB	9\$	0796
						50	01	000AD	SUBL3	4(SP), VBN_OFFSET, R0	0807
						56	12	000B0	MOVAB	-2(R0), VBN_OFFSET	0808
						04	AE	000B2	INCL	INDEX	0815
						FE	A0	000B7	BRW	1\$	0817
						0000'	CF	000BB	CLRL	R0	0817
						FF67	31	000BF	ADDL2	#12, SP	0817
						50	D4	000C2	POPR	#M<R2,R3,R4,R5,R6,R7>	
						0C	C0	000C4	RSB		
						00FC	8F	000C7			
						05	000CB				

; Routine Size: 204 bytes, Routine Base: \$CODE\$ + 01BE

```

: 825 0818 1 %SBTTL 'CHECK_LAST'
: 826 0819 1 GLOBAL ROUTINE RECL$$CHECK_LAST : RL$JSB_REG_8 =
: 827 0820 1 ++
: 828 0821 1
: 829 0822 1 Functional Description:
: 830 0823 1
: 831 0824 1 This routine checks to see if the current index record
: 832 0825 1 indexed by INDEX is the last record in the bucket and if it
: 833 0826 1 is the only record
: 834 0827 1
: 835 0828 1 Calling Sequence:
: 836 0829 1
: 837 0830 1 CHECK_LAST();
: 838 0831 1
: 839 0832 1 Input Parameters:
: 840 0833 1 none
: 841 0834 1
: 842 0835 1 Implicit Inputs:
: 843 0836 1
: 844 0837 1 BUCKET - address of buffer containing bucket
: 845 0838 1 INDEX - current index record (set by get_down_pointer)
: 846 0839 1
: 847 0840 1 Output Parameters:
: 848 0841 1
: 849 0842 1 None.
: 850 0843 1
: 851 0844 1 Implicit Outputs:
: 852 0845 1 none
: 853 0846 1
: 854 0847 1 Routine Value:
: 855 0848 1
: 856 0849 1 RECL$_SUCCESS - index record IS the last in bucket and there more
: 857 0850 1 then one record in the bucket
: 858 0851 1 RECL$_FAILURE - index record IS NOT the last in bucket or is the
: 859 0852 1 only one in the bucket
: 860 0853 1
: 861 0854 1 Routines Called:
: 862 0855 1
: 863 0856 1 None.
: 864 0857 1
: 865 0858 1 Side Effects:
: 866 0859 1
: 867 0860 1 None.
: 868 0861 1
: 869 0862 1 --
: 870 0863 1
: 871 0864 2 BEGIN
: 872 0865 2
: 873 0866 2 DEFINE_CTX;
: 874 0867 2 DEFINE_BUCKET;
: 875 0868 2 DEFINE_KEY_DESC;
: 876 0869 2 DEFINE_KEY_POINTER;
: 877 0870 2
: 878 0871 2 LOCAL
: 879 0872 2 VBN_OFFSET,
: 880 0873 2 LAST_VBN_OFFSET;
: 881 0874 2
```



```
: 882 0875 2 ! We can always reclaim the first record (even if its the last because the
: 883 0876 2 ! whole bucket will then be recalimed)
: 884 0877 2
: 885 0878 2 IF .INDEX EQL 0
: 886 0879 2 THEN
: 887 0880 2 RETURN RECL$_FAILURE;
: 888 0881 2
: 889 0882 2 ! Initialize offset in bucket to word containing VBN free space pointer
: 890 0883 2 ! so we can get the actual offset to the VBN free space.
: 891 0884 2
: 892 0885 2 VBN_OFFSET = .CTX [ CTX$_BUCKET_SIZE ] - 2 - 2;
: 893 0886 2
: 894 0887 2 ! Get actual offset of the last VBN (free_space pointer + 1)
: 895 0888 2
: 896 0889 2 LAST_VBN_OFFSET = .BUCKET [ .VBN_OFFSET, 0, 16, 0 ] + 1;
: 897 0890 2
: 898 0891 2 ! Now point to the current VBN down pointer found by get_down_pointer
: 899 0892 2
: 900 0893 2 VBN_OFFSET = .VBN_OFFSET -
: 901 0894 2 ( ( .BUCKET [ BKT$_PTR_SZ ] + 2 ) * ( .INDEX + 1 ) );
: 902 0895 2
: 903 0896 2 ! If they are equal then this is the last record in the bucket
: 904 0897 2
: 905 0898 2 IF .VBN_OFFSET EQLU .LAST_VBN_OFFSET
: 906 0899 2 THEN
: 907 0900 2 RETURN RECL$_SUCCESS
: 908 0901 2 ELSE
: 909 0902 2 RETURN RECL$_FAILURE
: 910 0903 2
: 911 0904 1 END;
```

			OC	BB	00000	RECL\$\$CHECK_LAST::			
						PUSRR	#*M<R2,R3>	0819	
		50	0000'	CF	D0	00002	MOVL	INDEX, R0	0878
				2A	13	00007	BEQL	1\$	
		53	58	AA	3C	00009	MOVZWL	88(CTX), VBN_OFFSET	0885
		53		03	C2	0000D	SUBL2	#3, VBN_OFFSET	
				7349	9F	00010	PUSHAB	-(VBN_OFFSET)[BUCKET]	0889
		52		9E	3C	00013	MOVZWL	@(SP)+, LAST_VBN_OFFSET	
				52	D6	00016	INCL	LAST_VBN_OFFSET	
		02		03	EF	00018	EXTZV	#3, #2, T3(BUCKET), R1	0894
		51		02	C0	0001E	ADDL2	#2, R1	
				50	D6	00021	INCL	R0	
		50		51	C4	00023	MULL2	R1, R0	
		53		50	C2	00026	SUBL2	R0, VBN_OFFSET	
		52		53	D1	00029	CMPL	VBN_OFFSET, LAST_VBN_OFFSET	0898
				05	12	0002C	BNEQ	1\$	
		50		01	D0	0002E	MOVL	#1, R0	0902
				02	11	00031	BRB	2\$	
				50	D4	00033	CLRL	R0	0904
				OC	BA	00035	POPR	#*M<R2,R3>	
					05	00037	RSB		

RECL\$REC  
V04-000

VAX-11 CONVERT/RECLAIM  
CHECK\_LAST

D 14  
15-Sep-1984 23:59:42  
14-Sep-1984 12:14:05

VAX-11 Bliss-32 V4.0-742  
DISK\$VMSMASTER:[CONV.SRC]RECLREC.B32;1 Page 26  
(8)

; Routine Size: 56 bytes, Routine Base: \$CODE\$ + 028A



```

: 913 0905 1 %SBTTL 'COMPARE_POINTER'
: 914 0906 1 GLOBAL ROUTINE RECL$$COMPARE_POINTER ( VBN ) : RL$JSB_REG_8 =
: 915 0907 1 ++
: 916 0908 1
: 917 0909 1 Functional Description:
: 918 0910 1
: 919 0911 1 This routine compares the next index record pointer in the current
: 920 0912 1 buffer for the specified down pointer if necessary is reads in the next
: 921 0913 1 bucket in the index chain to get the next index record.
: 922 0914 1
: 923 0915 1 Calling Sequence:
: 924 0916 1
: 925 0917 1 COMPARE_POINTER( VBN );
: 926 0918 1
: 927 0919 1 Input Parameters:
: 928 0920 1
: 929 0921 1 VBN - VBN to compare
: 930 0922 1
: 931 0923 1 Implicit Inputs:
: 932 0924 1
: 933 0925 1 BUCKET - address of buffer containing bucket
: 934 0926 1 INDEX - current index record (set by get_down_pointer)
: 935 0927 1
: 936 0928 1 Output Parameters:
: 937 0929 1
: 938 0930 1 None.
: 939 0931 1
: 940 0932 1 Implicit Outputs:
: 941 0933 1 none
: 942 0934 1
: 943 0935 1 Routine Value:
: 944 0936 1
: 945 0937 1 RECL$_SUCCESS - next index record DOES point to the vbn
: 946 0938 1 RECL$_FAILURE - next index record DOES NOT point to the vbn
: 947 0939 1
: 948 0940 1 Routines Called:
: 949 0941 1
: 950 0942 1 None.
: 951 0943 1
: 952 0944 1 Side Effects:
: 953 0945 1
: 954 0946 1 None.
: 955 0947 1
: 956 0948 1 --
: 957 0949 1
: 958 0950 2 BEGIN
: 959 0951 2
: 960 0952 2 DEFINE_CTX;
: 961 0953 2 DEFINE_BUCKET;
: 962 0954 2 DEFINE_KEY_DESC;
: 963 0955 2 DEFINE_KEY_POINTER;
: 964 0956 2
: 965 0957 2 LOCAL
: 966 0958 2 VBN_OFFSET,
: 967 0959 2 LAST_VBN_OFFSET,
: 968 0960 2 SEARCH_BUCKET : REF BLOCK [ ,BYTE ];
: 969 0961 2
```

```

: 970      0962 2
: 971      0963
: 972      0964
: 973      0965
: 974      0966
: 975      0967
: 976      0968
: 977      0969
: 978      0970
: 979      0971
: 980      0972
: 981      0973
: 982      0974
: 983      0975
: 984      0976
: 985      0977
: 986      0978
: 987      0979
: 988      0980
: 989      0981
: 990      0982
: 991      0983
: 992      0984
: 993      0985
: 994      0986
: 995      0987
: 996      0988
: 997      0989
: 998      0990
: 999      0991
1000      0992
1001      0993
1002      0994
1003      0995
1004      0996
1005      0997
1006      0998
1007      0999
1008      1000
1009      1001
1010      1002
1011      1003
1012      1004
1013      1005
1014      1006
1015      1007
1016      1008
1017      1009
1018      1010
1019      1011
1020      1012
1021      1013
1022      1014
1023      1015
1024      1016
1025      1017
: 1026      1018 4

! Initialize offset in bucket to word containing VBN free space pointer
! so we can get the actual offset to the VBN free space.
VBN_OFFSET = .CTX [ CTX$W_BUCKET_SIZE ] - 2 - 2;

! Get actual offset of the last VBN (free_space pointer + 1)
LAST_VBN_OFFSET = .BUCKET [ .VBN_OFFSET, 0, 16, 0 ] + 1;

! Now point to the current VBN down pointer found by get_down_pointer
VBN_OFFSET = .VBN_OFFSET -
              ( ( .BUCKET [ BKT$V_PTR_SZ ] + 2 ) * ( .INDEX + 1 ) );

! If this is not the end of the pointers then check the next vbn here
! else read in the next index bucket and search there
IF .VBN_OFFSET NEQU .LAST_VBN_OFFSET
THEN
  BEGIN
    ! Search in the current bucket
    SEARCH_BUCKET = .BUCKET;
    ! Point to the next vbn
    VBN_OFFSET = .VBN_OFFSET - ( .BUCKET [ BKT$V_PTR_SZ ] + 2 )
  END
ELSE
  ! Get the next bucket (if this is the last in the chain return failure)
  IF .BUCKET [ BKT$V_LASTBKT ]
  THEN
    RETURN RECL$_FAILURE
  ELSE
    BEGIN
      ! Search in the search buffer
      SEARCH_BUCKET = .RECL$GL_SEARCH_BUFFER;
      ! Read in the next index bucket
      CONV$AB_OUT_RAB [ RAB$L_UBF ] = .SEARCH_BUCKET;
      CONV$AB_OUT_RAB [ RAB$W_USZ ] = .CTX [ CTX$W_BUCKET_SIZE ];
      CONV$AB_OUT_RAB [ RAB$L_BKT ] = .BUCKET [ BKT$L_NXTBKT ];
      $READ( RAB=CONV$AB_OUT_RAB,ERR=CONV$$RMS_READ_ERROR );
      ! Point to the first vbn there
      VBN_OFFSET = .CTX [ CTX$W_BUCKET_SIZE ] - 2 - 2 -
                    ( .SEARCH_BUCKET [ BKT$V_PTR_SZ ] + 2 )
    END
  END

```



RECL\$REC  
V04-000

VAX-11 CONVERT/RECLAIM  
COMPARE\_POINTER

G 14  
15-Sep-1984 23:59:42  
14-Sep-1984 12:14:05

VAX-11 Bliss-32 V4.0-742  
DISK\$VMSMASTER:[CONV.SRC]RECLREC.B32;1 Page 29  
(9)

```

: 1027      1019 4
: 1028      1020 2
: 1029      1021 2
: 1030      1022 2
: 1031      1023 2
: 1032      1024 2
: 1033      1025 2
: 1034      1026 2
: 1035      1027 2
: 1036      1028 2
: 1037      1029 2
: 1038      1030 2
: 1039      1031 1

      END;
      ! Compare the vbns
      IF .VBN EQLU
        .SEARCH_BUCKET [ .VBN_OFFSET,0,((.SEARCH_BUCKET[ BKT$V_PTR_SZ ]+2)*8),0 ]
      THEN
        RETURN RECL$_SUCCESS
      ELSE
        RETURN RECL$_FAILURE
      END;
```

.EXTRN SYSS\$READ

```

      1C BB 00000 RECL$_COMPARE_POINTER::
      54 58 AA 3C 00002 PUSHR #M<R2,R3,R4> 0906
      52 74 DE 00006 MOVZWL 88(CTX), R4 0966
      6249 9F 00009 MOVAL -(R4), VBN_OFFSET
      53 9E 3C 0000C PUSHAB (VBN_OFFSET)[BUCKET] 0970
      53 96 0000F MOVZWL @ (SP)+, LAST_VBN_OFFSET
      02 03 E 00011 INCL LAST_VBN_OFFSET
      51 02 C0 00017 EXTZV #3, #2, T3(BUCKET), R1 0975
      50 01 C1 0001A ADDL2 #2, R1
      52 51 C4 00020 ADDL3 #1, INDEX, R0
      53 50 C2 00023 MULL2 R1, R0
      53 52 D1 00026 SUBL2 R0, VBN_OFFSET
      53 08 13 00029 CMPL VBN_OFFSET, LAST_VBN_OFFSET 0980
      53 59 D0 0002B BEQL 1$
      52 51 C2 0002E MOVL BUCKET, SEARCH_BUCKET 0986
      37 11 00031 SUBL2 R1, VBN_OFFSET 0990
      50 0D A9 E8 00033 BRB 2$
      53 0000G CF D0 00037 BLBS 13(BUCKET), 3$ 0997
      0000G CF 53 D0 0003C MOVL RECL$GL_SEARCH_BUFFER, SEARCH_BUCKET 1005
      0000G CF 58 AA B0 00041 MOVL SEARCH_BUCKET, CONV$AB_OUT_RAB+36 1009
      0000G CF 08 A9 D0 00047 MOVW 88(CTX), CONV$AB_OUT_RAB+32 1010
      0000G CF 9F 0004D MOVL 8(BUCKET), CONV$AB_OUT_RAB+56 1011
      0000G CF 9F 00051 PUSHAB CONV$SRMS_READ_ERROR 1013
      00 02 FB 00055 PUSHAB CONV$AB_OUT_RAB
      50 03 E 0005C CALLS #2, SYSS$READ
      54 50 C3 00062 EXTZV #3, #2, 13(SEARCH_BUCKET), R0 1018
      52 FE A0 9E 00066 SUBL3 R0, R4, R0
      50 08 C4 00070 MOVAB -2(R0), VBN_OFFSET 1017
      50 10 C0 00073 EXTZV #3, #2, 13(SEARCH_BUCKET), R0 1025
      51 50 00 EF 00076 ADDL2 #16, R0
      51 10 AE D1 0007C EXTZV #0, R0, (VBN_OFFSET)[SEARCH_BUCKET], R1
      50 05 12 00080 CMPL VBN, R1
      01 D0 00082 BNEQ 3$
      02 11 00085 MOVL #1, R0 1029
      50 D4 00087 BRB 4$
      1C BA 00089 CLRL R0 1031
      05 0008B POPR #M<R2,R3,R4>
      RSB
```

RECL\$REC  
V04-000

VAX-11 CONVERT/RECLAIM  
COMPARE\_POINTER

H 14  
15-Sep-1984 23:59:42  
14-Sep-1984 12:14:05

VAX-11 Bliss-32 V4.0-742  
DISK\$VMSMASTER:[CONV.SRC]RECLREC.B32;1 Page 30  
(9)

; Routine Size: 140 bytes, Routine Base: \$CODE\$ + 02C2



```
1041 1032 1 %SBTTL 'SWING_POINTER'
1042 1033 1 GLOBAL ROUTINE RECL$SWING_POINTER ( VBN ) : RL$JSB_REG_8 NOVALUE =
1043 1034 1 ++
1044 1035 1
1045 1036 1 Functional Description:
1046 1037 1
1047 1038 1 This routine will stuff the VBN into the curretn index record
1048 1039 1
1049 1040 1 Calling Sequence:
1050 1041 1
1051 1042 1 SWING_POINTER( VBN );
1052 1043 1
1053 1044 1 Input Parameters:
1054 1045 1
1055 1046 1 VBN - VBN to stuff
1056 1047 1
1057 1048 1 Implicit Inputs:
1058 1049 1
1059 1050 1 BUCKET - address of buffer containing bucket
1060 1051 1 INDEX - index record to stuff
1061 1052 1
1062 1053 1 Output Parameters:
1063 1054 1
1064 1055 1 None.
1065 1056 1
1066 1057 1 Implicit Outputs:
1067 1058 1 none
1068 1059 1
1069 1060 1 Routine Value:
1070 1061 1 none
1071 1062 1
1072 1063 1 Routines Called:
1073 1064 1
1074 1065 1 None.
1075 1066 1
1076 1067 1 Side Effects:
1077 1068 1
1078 1069 1 None.
1079 1070 1
1080 1071 1 --
1081 1072 1
1082 1073 2 BEGIN
1083 1074 2
1084 1075 2 DEFINE_CTX;
1085 1076 2 DEFINE_BUCKET;
1086 1077 2 DEFINE_KEY_DESC;
1087 1078 2 DEFINE_KEY_POINTER;
1088 1079 2
1089 1080 2 LOCAL
1090 1081 2 VBN_OFFSET;
1091 1082 2
1092 1083 2 ! Point to current VBN down pointer
1093 1084 2 ! Which is: Bucket size - 2 bytes for check and spare - 2 bytes for
1094 1085 2 ! vbn freespace pointer - index into the array
1095 1086 2
1096 1087 2 VBN_OFFSET = .CTX [ CTX$W BUCKET SIZE ] - 2 - 2 -
1097 1088 2 ( .BUCKET [ BKT$V_PTR_SZ ] + 2 ) * ( .INDEX + 1 );
```

RECL\$REC  
V04-000

VAX-11 CONVERT/RECLAIM  
SWING\_POINTER

J 14  
15-Sep-1984 23:59:42  
14-Sep-1984 12:14:05

VAX-11 Bliss-32 V4.0-742  
DISK\$VMSMASTER:[CONV.SRC]RECLREC.B32;1 (10)  
Page 32

```
: 1098      1089  2
: 1099      1090  2      ! Stuff the vbn
: 1100      1091  2      !
: 1101      1092  2      BUCKET [ .VBN_OFFSET,0,( ( .BUCKET [ BKT$V_PTR_SZ ] + 2 ) * 8 ),0 ] = .VBN;
: 1102      1093  2
: 1103      1094  2      RETURN
: 1104      1095  2
: 1105      1096  1      END;
```

				0C	BB	00000	RECL\$\$\$SWING_POINTER::		
							PUSHR	#^M<R2,R3>	: 1033
52	0D	A9	02	03	EF	00002	EXTZV	#3, #2, 13(BUCKET), R2	: 1088
			51	A2	9E	00008	MOVAB	2(R2), R1	
		50	CF	01	C1	0000C	ADDL3	#1, INDEX, R0	
			50	51	C4	00012	MULL2	R1, R0	
			53	AA	3C	00015	MOVZWL	88(CTX), R3	
		50	53	50	C3	00019	SUBL3	R0, R3, R0	
			50	03	C2	0001D	SUBL2	#3, VBN_OFFSET	: 1087
		51	52	03	78	00020	ASHL	#3, R2, R1	: 1092
			51	10	C0	00024	ADDL2	#16, R1	
7049		51	00	0C	AE	F0 00027	INSV	VBN, #0, R1, -(VBN_OFFSET)[BUCKET]	
				0C	BA	0002E	POPR	#^M<R2,R3>	: 1096
					05	00030	RSB		:

: Routine Size: 49 bytes,      Routine Base: \$CODE\$ + 034E



```
1107 1097 1 %SBTTL 'REMOVE INDEX RECORD'
1108 1098 1 GLOBAL ROUTINE RECL$$REMOVE_INDEX_RECORD : RLSJSB_REG_8 NOVALUE =
1109 1099 1 ++
1110 1100 1
1111 1101 1 Functional Description:
1112 1102 1
1113 1103 1 This routine actually squishes out the index record from the index
1114 1104 1 bucket.
1115 1105 1
1116 1106 1 Calling Sequence:
1117 1107 1
1118 1108 1 REMOVE_INDEX_RECORD();
1119 1109 1
1120 1110 1 Input Parameters:
1121 1111 1
1122 1112 1 None.
1123 1113 1
1124 1114 1 Implicit Inputs:
1125 1115 1
1126 1116 1 INDEX - number of the index record to remove
1127 1117 1 KEY_POINTER - points to key part of index record to remove
1128 1118 1 KEY_BUFFER_1 - contains fully expanded previous key
1129 1119 1 KEY_BUFFER_2 - contains fully expanded current key
1130 1120 1 BUCKET - points to buffer containing bucket
1131 1121 1
1132 1122 1 Output Parameters:
1133 1123 1
1134 1124 1 None.
1135 1125 1
1136 1126 1 Implicit Outputs:
1137 1127 1
1138 1128 1 Index bucket has more freespace, since a record was squished out.
1139 1129 1
1140 1130 1 Routine Value:
1141 1131 1
1142 1132 1 None.
1143 1133 1
1144 1134 1 Routines Called:
1145 1135 1
1146 1136 1 RECOMPRESS_RECORD
1147 1137 1
1148 1138 1 Side Effects:
1149 1139 1
1150 1140 1 None.
1151 1141 1
1152 1142 1 --
1153 1143 1
1154 1144 2 BEGIN
1155 1145 2
1156 1146 2 DEFINE_CTX;
1157 1147 2 DEFINE_BUCKET;
1158 1148 2 DEFINE_KEY_DESC;
1159 1149 2 DEFINE_KEY_POINTER;
1160 1150 2
1161 1151 2 ++
1162 1152 2
1163 1153 2 ! Squish out the VBN part of the index record
```

```

1164      1154      !
1165      1155      !--
1166      1156
1167      1157      BEGIN
1168      1158
1169      1159      LOCAL
1170      1160          OFFSET,           ! Offset to the vbn freespace pointer
1171      1161          VBN_SIZE,       ! Size of vbn in bytes
1172      1162          BITS,         ! Size of vbn in bits
1173      1163          FREESPACE,     ! Pointer offset to the top of the vbns
1174      1164          VBN,         ! Pointer offset to the vbn to remove
1175      1165          SOURCE,       ! Pointer offset to the Source
1176      1166          DEST;        ! Pointer offset to the Destination
1177      1167
1178      1168      ! Find the offset to the vbn freespace pointer
1179      1169      OFFSET = .CTX [ CTX$W_BUCKET_SIZE ] - 4;
1180      1170
1181      1171      ! Get the size of the vbns in bytes
1182      1172
1183      1173      VBN_SIZE = .BUCKET [ BKT$V_PTR_SZ ] + 2;
1184      1174
1185      1175
1186      1176      ! Now get it in bits
1187      1177
1188      1178      BITS = .VBN_SIZE * 8;
1189      1179
1190      1180      ! Find the top the vbns
1191      1181
1192      1182      FREESPACE = .BUCKET [ .OFFSET,0,16,0 ];
1193      1183
1194      1184      ! Find the vbn we want to remove
1195      1185
1196      1186      VBN = .OFFSET - ( .VBN_SIZE * ( .INDEX + 1 ) );
1197      1187
1198      1188      ! Set up the destindtion
1199      1189
1200      1190      DEST = .VBN;
1201      1191
1202      1192      ! And the source
1203      1193
1204      1194      SOURCE = .DEST - .VBN_SIZE;
1205      1195
1206      1196      ! Do each vbn
1207      1197
1208      1198      WHILE .SOURCE GEQU .FREESPACE
1209      1199      DO
1210      1200          BEGIN
1211      1201              ! Copy the vbn to the new location
1212      1202              BUCKET [ .DEST,0,.BITS,0 ] = .BUCKET [ .SOURCE,0,.BITS,0 ];
1213      1203
1214      1204              ! Update the pointers
1215      1205
1216      1206              DEST = .DEST - .VBN_SIZE;
1217      1207              SOURCE = .SOURCE - .VBN_SIZE
1218      1208
1219      1209
1220      1210

```



```
1221      END;
1222
1223      ! Update the freespace pointer in the bucket
1224      BUCKET [ .OFFSET,0,16,0 ] = .FREESPACE + .VBN_SIZE;
1225
1226      ! If freespace pointer points to the bottom of the bucket it is
1227      ! empty so don't bother to fool with the data part (but do set
1228      ! the keyfreespace pointer)
1229      IF .BUCKET [ .OFFSET,0,16,0 ] EQLU .OFFSET
1230      THEN
1231      BEGIN
1232      BUCKET[ BKT$W_KEYFRESPC ] = BKT$C_OVERHDSZ;
1233      RETURN
1234      END
1235
1236      END;
1237
1238      !++
1239      ! Squeeze out the KEY part of the index record
1240      !--
1241
1242      BEGIN
1243      LOCAL
1244      DELETE_SIZE;
1245
1246      ! Calculate from address and size for squish differently if index is
1247      ! compressed or not. Also do KEYFRESPC depending on index compression.
1248      IF .KEY_DESC[ KEY$V_IDX_COMPR ]
1249      THEN
1250      BEGIN
1251      LOCAL
1252      NEXT          : REF BLOCK [ .BYTE ]; ! Pointer to the next key
1253                                     ! to replace the deleted one
1254
1255      ! The size of the deleted space is size of the old record MINUS
1256      ! the DIFFERENCE between the size of next record before compression
1257      ! and the size of it after compression.
1258      ! First save the size of old record.
1259      DELETE_SIZE = .KEY_POINTER [ KEYSB_LENGTH ] + 2;
1260
1261      ! Next thing to do is recompress the next record after the current
1262      ! one we start by coping it into key_buffer_2 (where the to-be-deleted
1263      ! key is)
1264      NEXT = .KEY_POINTER + .KEY_POINTER [ KEYSB_LENGTH ] + 2;
1265
1266      ! If there IS a next key then copy it and compress it
1267      IF .NEXT LSSU ( .BUCKET + .BUCKET [ BKT$W_KEYFRESPC ] )
```

RECL\$REC  
V04-000

VAX-11 CONVERT/RECLAIM  
REMOVE\_INDEX\_RECORD

N 14  
15-Sep-1984 23:59:42  
14-Sep-1984 12:14:05

VAX-11 Bliss-32 V4.0-742  
DISK\$VMSMASTER:[CONV.SRC]RECLREC.B32;1 (11) Page 36

```

1278      1268      4      THEN
1279      1269      5      BEGIN
1280      1270      5      LOCAL OLD_SIZE;
1281      1271      5      ! Save the old size of the next record
1282      1272      5      !
1283      1273      5      ! Save the old size of the next record
1284      1274      5      !
1285      1275      5      OLD_SIZE = .NEXT [ KEYSB_LENGTH ];
1286      1276      5      ! Copy the next key while expanding the rear
1287      1277      5      !
1288      1278      5      CH$COPY( .NEXT [ KEYSB_LENGTH ],
1289      1279      5      .NEXT + 2,
1290      1280      5      ( .NEXT + 1 + .NEXT [ KEYSB_LENGTH ] ),
1291      1281      5      .KEY_DESC [ KEYSB_KEYSZ ] - .NEXT [ KEYSB_FRONT_COUNT ],
1292      1282      5      KEY_BUFFER_2 + 2 + .NEXT [ KEYSB_FRONT_COUNT ] );
1293      1283      5      !
1294      1284      5      ! Recompress the new key in key_buffer_2
1295      1285      5      !
1296      1286      5      RECOMPRESS_RECORD();
1297      1287      5      !
1298      1288      5      ! Key_buffer_2 now contains a compressed key (w/control info)
1299      1289      5      ! so move it into the bucket
1300      1290      5      !
1301      1291      5      CH$MOVE( .KEY_BUFFER_2 [ KEYSB_LENGTH ] + 2,
1302      1292      5      KEY_BUFFER_2,
1303      1293      5      .KEY_POINTER );
1304      1294      5      !
1305      1295      5      ! Now we can figure the amount of space deleted
1306      1296      5      !
1307      1297      5      DELETE_SIZE = .DELETE_SIZE -
1308      1298      5      ( .KEY_POINTER [ KEYSB_LENGTH ] - .OLD_SIZE );
1309      1299      5      !
1310      1300      5      ! We must now move the rest of the keys in the bucket
1311      1301      5      !
1312      1302      5      CH$MOVE( ( .BUCKET + .BUCKET [ BKTSW_KEYFRESPC ] ) -
1313      1303      5      ( .NEXT + .NEXT [ KEYSB_LENGTH ] + 2 ),
1314      1304      5      .NEXT + .NEXT [ KEYSB_LENGTH ] + 2,
1315      1305      5      .KEY_POINTER + .KEY_POINTER [ KEYSB_LENGTH ] + 2 );
1316      1306      5      !
1317      1307      5      !
1318      1308      5      END;
1319      1309      4      END
1320      1310      4      ELSE
1321      1311      4      BEGIN
1322      1312      4      ! Set the delete size
1323      1313      4      !
1324      1314      4      DELETE_SIZE = .KEY_DESC [ KEYSB_KEYSZ ];
1325      1315      4      !
1326      1316      4      ! Move the rest of the keys
1327      1317      4      !
1328      1318      4      !
1329      1319      4      CH$MOVE( ( .BUCKET + .BUCKET [ BKTSW_KEYFRESPC ] ) -
1330      1320      4      ( .KEY_POINTER + .KEY_DESC [ KEYSB_KEYSZ ] ),
1331      1321      4      .KEY_POINTER + .KEY_POINTER [ KEYSB_LENGTH ] + 2,
1332      1322      4      .KEY_POINTER );
1333      1323      4      !
1334      1324      4      !

```



```
1335      1325      3      END;  
1336      1326      3  
1337      1327      3      ! Update KEYFRESPC since we squished out a key  
1338      1328      3  
1339      1329      3      BUCKET [ BKT$W_KEYFRESPC ] = .BUCKET [ BKT$W_KEYFRESPC ] - .DELETE_SIZE  
1340      1330      3  
1341      1331      3      END;  
1342      1332      3  
1343      1333      3      RETURN  
1344      1334      3  
1345      1335      1      END;
```

```
00FC 8F BB 00000 RECL$REMOVE INDEX RECORD::  
5E      04 C2 00004 PUSHB #M<R2,R3,R4,R5,R6,R7> 1098  
51      58 AA 3C 00007 SUBL2 #4, SP 1170  
51      03 C2 0000B MOVZWL 88(CTX), OFFSET 1174  
52      02 EF 0000E SUBL2 #3, OFFSET 1174  
52      02 C0 00014 EXTZV #3, #2, 13(BUCKET), VBN_SIZE 1178  
52      03 78 00017 ADDL2 #2, VBN_SIZE 1182  
7149 9F 0001B ASHL #3, VBN_SIZE, BITS 1186  
9E 3C 0001E PUSHAB -(OFFSET)[BUCKET] 1190  
01 C1 00021 MOVZWL @ (SP)+, FREESPACE 1194  
52 C4 00027 ADDL3 #1, INDEX, R0 1198  
50 C3 0002A MULL2 VBN_SIZE, R0 1204  
50 D0 0002E SUBL3 R0, -OFFSET, VBN 1208  
53 52 C3 00031 MOVL VBN, DEST 1209  
53 54 C3 00035 SUBL3 VBN_SIZE, DEST, SOURCE 1215  
55 54 D1 00038 CMPL SOURCE, FREESPACE 1221  
14 1F 00038 BLSSU 2$ 1224  
00 EF 0003A EXTZV #0, BITS, (SOURCE)[BUCKET], R0 1223  
50 F0 00040 INSV R0, #0, BITS, (DEST)[BUCKET] 1244  
52 C2 00046 SUBL2 VBN_SIZE, DEST 1257  
52 C2 00049 SUBL2 VBN_SIZE, SOURCE 1263  
E7 11 0004C BRB 1$ 1267  
6149 9F 0004E PUSHAB (OFFSET)[BUCKET] 1275  
52 A1 00051 ADDW3 VBN_SIZE, FREESPACE, @ (SP)+ 1281  
00 ED 00055 CMPZV #0, #16, (OFFSET)[BUCKET], OFFSET 1282  
07 12 0005B BNEQ 3$  
0E B0 0005D MOVW #14, 4(BUCKET)  
0091 31 00061 BRW 6$  
03 E1 00064 BBC #3, 16(KEY_DESC), 4$  
68 9A 00069 MOVZBL (KEY_POINTER), R0  
02 A0 9E 0006C MOVAB 2(R0), DELETE_SIZE  
02 A048 9E 00070 MOVAB 2(R0)[KEY_POINTER], NEXT  
04 A9 3C 00075 MOVZWL 4(BUCKET), R0  
59 C0 00079 ADDL2 BUCKET, R0  
51 D1 0007C CMPL NEXT, R0  
70 1E 0007F BGEQU 5$  
61 9A 00081 MOVZBL (NEXT), R3  
53 D0 00084 MOVL R3, OLD_SIZE  
51 53 C1 00087 ADDL3 R3, NEXT, (SP)  
01 A1 9A 0008B MOVZBL 1(NEXT), R0  
14 AB 9A 0008F MOVZBL 20(KEY_DESC), R2
```

RECL\$REC  
V04-000

VAX-11 CONVERT/RECLAIM  
REMOVE\_INDEX\_RECORD

C 15  
15-Sep-1984 23:59:42  
14-Sep-1984 12:14:05

VAX-11 Bliss-32 V4.0-742  
DISK\$VMSMASTER:[CONV.SRC]RECLREC.B32;1  
Page 38  
(11)

52	7E	02	52	50	C2	00093	SUBL2	R0, R2	:	1283	
	9E		6E	01	C1	00096	ADDL3	#1, (SP), -(SP)	:		
			A1	53	2C	0009A	MOVC5	R3, 2(NEXT), @ (SP)+, R2, KEY_BUFFER_2+2[R0]	:		
				0000	CF40	000A0			:		
				0000	0000V	30	000A4	BSBW	RECOMPRESS_RECORD	:	1287
			50	CF	9A	000A7	MOVZBL	KEY_BUFFER_2, R0	:	1292	
			50	02	C0	000AC	ADDL2	#2, R0	:		
	68	0000	50	50	28	000AF	MOVC3	R0, KEY_BUFFER_2, (KEY_POINTER)	:	1294	
			51	68	9A	000B5	MOVZBL	(KEY_POINTER), R1	:	1299	
			56	51	C2	000B8	SUBL2	R1, R6	:		
			57	56	C0	000BB	ADDL2	R6, DELETE_SIZE	:		
			50	04	A9	3C	000BE	MOVZWL	4(BUCKET), R0	:	1303
			50	59	C0	000C2	ADDL2	BUCKET, R0	:		
			50	6E	C2	000C5	SUBL2	(SP), R0	:	1304	
			50	02	C2	000C8	SUBL2	#2, R0	:	1303	
			52	6E	D0	000CB	MOVL	(SP), R2	:	1305	
	02 A148	02	A2	50	28	000CE	MOVC3	R0, 2(R2), 2(R1)[KEY_POINTER]	:	1306	
				1A	11	000D5	BRB	5\$	:	1244	
			52	14	AB	9A	000D7	MOVZBL	20(KEY_DESC), R2	:	1316
			57	52	D0	000DB	MOVL	R2, DELETE_SIZE	:		
			50	04	A9	3C	000DE	MOVZWL	4(BUCKET), R0	:	1320
	51		59	50	C1	000E2	ADDL3	R0, BUCKET, R1	:		
	50		58	52	C1	000E6	ADDL3	R2, KEY_POINTER, R0	:	1321	
			51	50	C2	000EA	SUBL2	R0, R1	:		
	68		60	51	28	000ED	MOVC3	R1, (R0), (KEY_POINTER)	:	1323	
		04	A9	57	A2	000F1	SUBW2	DELETE_SIZE, 4(BUCKET)	:	1329	
			5E	04	C0	000F5	ADDL2	#4, SP	:	1335	
				00FC	8F	BA	000F8	POPR	#M<R2,R3,R4,R5,R6,R7>	:	
					05	000FC	RSB		:		

; Routine Size: 253 bytes, Routine Base: \$CODE\$ + 037F

; 1346 1336 1



```
: 1348 1337 1 %SBTTL 'RECOMPRESS_RECORD'
: 1349 1338 1 ROUTINE RECOMPRESS_RECORD : RL$JSB_REG_8 NOVALUE =
: 1350 1339 1 ++
: 1351 1340 1
: 1352 1341 1 Functional Description:
: 1353 1342 1
: 1354 1343 1 This routine will recompress the index record in key_buffer_2
: 1355 1344 1
: 1356 1345 1 Calling Sequence:
: 1357 1346 1
: 1358 1347 1 RECOMPRESS_RECORD()
: 1359 1348 1
: 1360 1349 1 Input Parameters:
: 1361 1350 1
: 1362 1351 1 None.
: 1363 1352 1
: 1364 1353 1 Implicit Inputs:
: 1365 1354 1
: 1366 1355 1 KEY_BUFFER_1 - contains expanded key to base re-compression upon
: 1367 1356 1 KEY_BUFFER_2 - contains expanded key to re-compress
: 1368 1357 1
: 1369 1358 1 Output Parameters:
: 1370 1359 1
: 1371 1360 1 None.
: 1372 1361 1
: 1373 1362 1 Implicit Outputs:
: 1374 1363 1
: 1375 1364 1 None.
: 1376 1365 1
: 1377 1366 1 Routine Value:
: 1378 1367 1
: 1379 1368 1 None.
: 1380 1369 1
: 1381 1370 1 Routines Called:
: 1382 1371 1
: 1383 1372 1 None.
: 1384 1373 1
: 1385 1374 1 Side Effects:
: 1386 1375 1
: 1387 1376 1 Index record in key_buffer_2 is compressed.
: 1388 1377 1
: 1389 1378 1 --
: 1390 1379 1
: 1391 1380 2 BEGIN
: 1392 1381 2
: 1393 1382 2 DEFINE_CTX;
: 1394 1383 2 DEFINE_BUCKET;
: 1395 1384 2 DEFINE_KEY_DESC;
: 1396 1385 2 DEFINE_KEY_POINTER;
: 1397 1386 2
: 1398 1387 2 BIND
: 1399 1388 2 KEY_1 = KEY_BUFFER_1 + 2 : VECTOR [ ,BYTE ], ! Key part of the record
: 1400 1389 2 KEY_2 = KEY_BUFFER_2 + 2 : VECTOR [ ,BYTE ];
: 1401 1390 2
: 1402 1391 2 LOCAL
: 1403 1392 2 LENGTH;
: 1404 1393 2
```

```
: 1405      1394 2      ! Assume no compression
: 1406      1395 2      !
: 1407      1396 2      KEY_BUFFER_2 [ KEY$B_FRONT_COUNT ] = 0;
: 1408      1397 2      !
: 1409      1398 2      LENGTH = .KEY_DESC [ KEY$B_KEYSZ ];
: 1410      1399 2      !
: 1411      1400 2      ! If this is NOT the first key in the bucket do front compression
: 1412      1401 2      !
: 1413      1402 2      IF .INDEX NEQU 0
: 1414      1403 2      THEN
: 1415      1404 2          ! Find the first position where the two keys differ
: 1416      1405 2          !
: 1417      1406 2          ! Find the first position where the two keys differ
: 1418      1407 2          INCR I FROM 0 TO ( .LENGTH - 1 ) BY 1
: 1419      1408 2          DO
: 1420      1409 2              ! If the characters are not equal we found the end
: 1421      1410 2              !
: 1422      1411 2              ! If ( .KEY_1 [ .I ] NEQU .KEY_2 [ .I ] )
: 1423      1412 2              IF ( .KEY_1 [ .I ] NEQU .KEY_2 [ .I ] )
: 1424      1413 2              THEN
: 1425      1414 2                  BEGIN
: 1426      1415 2                      ! I is now the number of compressed characters
: 1427      1416 2                      !
: 1428      1417 2                      ! I is now the number of compressed characters
: 1429      1418 2                      KEY_BUFFER_2 [ KEY$B_FRONT_COUNT ] = .I;
: 1430      1419 2                      !
: 1431      1420 2                      ! Shorten the length
: 1432      1421 2                      !
: 1433      1422 2                      LENGTH = .LENGTH - .I;
: 1434      1423 2                      !
: 1435      1424 2                      ! If there was some compression move the key a little
: 1436      1425 2                      !
: 1437      1426 2                      IF .I NEQU 0
: 1438      1427 2                      THEN
: 1439      1428 2                          CH$MOVE( .LENGTH, KEY_2 + .I ,KEY_2 );
: 1440      1429 2                      !
: 1441      1430 2                      EXITLOOP
: 1442      1431 2                      !
: 1443      1432 2                      END;
: 1444      1433 2          !
: 1445      1434 2          ! Do rear end truncation
: 1446      1435 2          !
: 1447      1436 2          WHILE .LENGTH GTRU 1
: 1448      1437 2          DO
: 1449      1438 2              ! If the trailing characters are the same cut it short
: 1450      1439 2              !
: 1451      1440 2              ! If .KEY_2 [ .LENGTH - 1 ] EQLU .KEY_2 [ .LENGTH - 2 ]
: 1452      1441 2              IF .KEY_2 [ .LENGTH - 1 ] EQLU .KEY_2 [ .LENGTH - 2 ]
: 1453      1442 2              THEN
: 1454      1443 2                  LENGTH = .LENGTH - 1
: 1455      1444 2              ELSE
: 1456      1445 2                  EXITLOOP;
: 1457      1446 2          !
: 1458      1447 2          ! Set the length field
: 1459      1448 2          !
: 1460      1449 2          KEY_BUFFER_2 [ KEY$B_LENGTH ] = .LENGTH;
: 1461      1450 2          !
```



RECL\$REC  
V04-000

VAX-11 CONVERT/RECLAIM  
RECOMPRESS\_RECORD

F 15  
15-Sep-1984 23:59:42  
14-Sep-1984 12:14:05

VAX-11 BLISS-32 V4.0-742  
DISK\$VMSMASTER:[CONV.SRC]RECLREC.B32;1 (12) Page 41

: 1462  
: 1463  
: 1464  
1451 2 RETURN  
1452 2  
1453 1 END;

KEY\_1=  
KEY\_2=

KEY\_BUFFER\_1+2  
KEY\_BUFFER\_2+2

		00FC	8F	BB	00000	RECOMPRESS_RECORD:	
	5E		04	C2	00004	PUSHR	#^M<R2,R3,R4,R5,R6,R7>
		0000'	CF	94	00007	SUBL2	#4, SP
	56	14	AB	9A	0000B	CLRB	KEY_BUFFER_2+1
		0000'	CF	D5	0000F	MOVZBL	20(KEY_DESC), LENGTH
			2E	13	00013	TSTL	INDEX
	6E		56	D0	00015	BEQL	3\$
	57		01	CE	00018	MOVL	LENGTH, (SP)
			22	11	0001B	MNEGL	#1, I
	0000'CF47	0000'CF47	91	0001D	1\$:	BRB	2\$
			17	13	00026	CMPB	KEY_1[I], KEY_2[I]
	0000' CF		57	90	00028	BEQL	2\$
	56		57	C2	0002D	MOVB	I, KEY_BUFFER_2+1
			57	D5	00030	SUBL2	I, LENGTH
			0F	13	00032	TSTL	I
	0000' CF	0000'CF47	56	28	00034	BEQL	3\$
			04	11	0003D	MOVC3	LENGTH, KEY_2[I], KEY_2
	DA	57	6E	F2	0003F	BRB	3\$
	01		56	D1	00043	AOBLS	(SP), I, 1\$
			0F	1B	00046	CMPL	LENGTH, #1
	0000'CF46	0000'CF46	91	00048	3\$:	BLEQU	4\$
			04	12	00051	CMPB	KEY_2-1[LENGTH], KEY_2-2[LENGTH]
			56	D7	00053	BNEQ	4\$
			EC	11	00055	DECL	LENGTH
	0000' CF		56	90	00057	BRB	3\$
	5E		04	C0	0005C	MOVB	LENGTH, KEY_BUFFER_2
		00FC	8F	BA	0005F	ADDL2	#4, SP
			05	00063	POPR	#^M<R2,R3,R4,R5,R6,R7>	
					RSB		

: Routine Size: 100 bytes, Routine Base: \$CODE\$ + 047C

: 1465  
: 1466  
1454 1  
1455 0 END ELUDOM

# PSECT SUMMARY

Name	Bytes	Attributes
\$OWNS	529	NOVEC, WRT, RD, NOEXE, NOSHR, LCL, REL, CON, NOPIC, ALIGN(2)
\$CODE\$	1248	NOVEC, NOWRT, RD, EXE, NOSHR, LCL, REL, CON, NOPIC, ALIGN(2)

RECL\$REC  
V04-000

VAX-11 CONVERT/RECLAIM  
RECOMPRESS\_RECORD

G 15  
15-Sep-1984 23:59:42  
14-Sep-1984 12:14:05

VAX-11 Bliss-32 V4.0-742  
DISK\$VM\$MASTER:[CONV.SRC]RECLREC.B32;1 (12) Page 42

Library Statistics

File	----- Total	Symbols Loaded	----- Percent	Pages Mapped	Processing Time
_\$255\$DUA28:[SYSLIB]LIB.L32;1	18619	34	0	1000	00:01.9
_\$255\$DUA28:[CONV.SRC]CONVERT.L32;1	165	11	6	17	00:00.2

COMMAND QUALIFIERS

BLISS/CHECK=(FIELD,INITIAL,OPTIMIZE)/LIS=LIS\$:RECLREC/OBJ=OBJ\$:RECLREC MSRC\$:RECLREC/UPDATE=(ENH\$:RECLREC)

: Size: 1248 code + 529 data bytes  
: Run Time: 00:30.1  
: Elapsed Time: 01:45.3  
: Lines/CPU Min: 2903  
: Lexemes/CPU-Min: 15252  
: Memory Used: 148 pages  
: Compilation Complete



0066 AH-BT13A-SE  
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION  
CONFIDENTIAL AND PROPRIETARY

